Presented at EURESCO99, Problem-Solving Environments: Infrastructure and Prototypes, San Feliu de Guixols, Spain, 12-17 June 1999

Automated Techniques for Performance Evaluation of Parallel Systems[1]

Stéphane Simon, Michel Courson and Alan Mink

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8951, USA
michel.courson@nist.gov
(301) 975-5686 (office) -- (301) 869 7429 (Fax)

Institut National des Télécommunications
5, rue Charles Fourier
91011 Evry Cédex, France
stephane.simon@int-evry.fr

Abstract

Benchmarking and performance evaluation of high performance computing are a continuously on-going process that consumes significant staff time and generates large amounts of data to be stored and analyzed. The goal of this work is to propose an automated method to generate, capture and analyze an extensive performance profile. Although our initial efforts focus on commodity clusters, they are applicable to any parallel or distributed high performance computing system. Such an automated system has three main components: (1) a data collection and storage module, (2) a data analysis module, and (3) a run-time execution module. The focus of this report is on the data collection and storage module.

Keywords: cluster computing, profiling, performance analysis, database, SNMP.

[1] Certain commercial items may be identified but that does not imply recommendation or endorsement by NIST, nor does it imply that those items are necessarily the best available for the purpose.

Table of contents

1 Presentation

1.1 The Cluster Profiling Project

The Scalable Parallel Systems and Applications Group focuses on research and advanced development of measurement techniques for high performance parallel computing, including cluster environments.

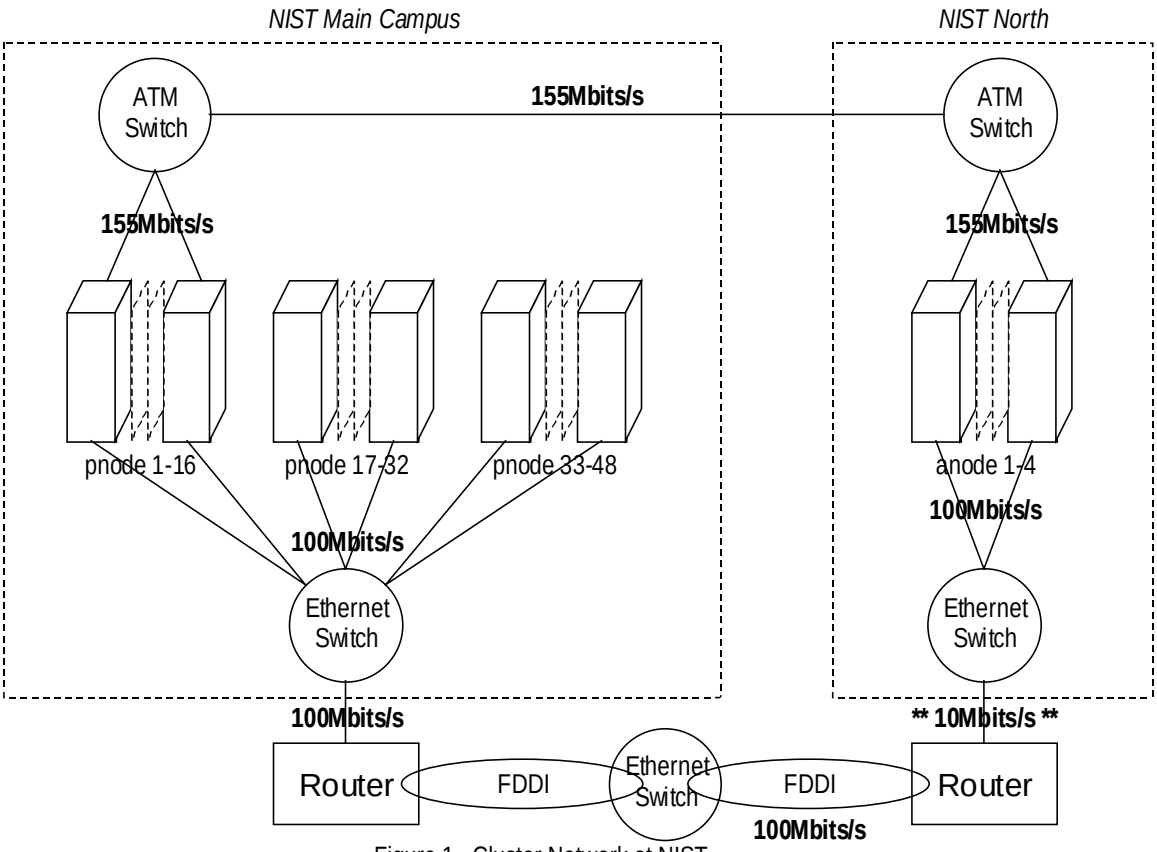

Figure 1. Cluster Network at NIST.

Commodity clusters, built around low-cost ordinary PCs, provide an inexpensive and scalable solution for scientific computing. We have built several experimental PC clusters interconnected with both ATM and Fast-Ethernet, some at different physical locations on the campus, allowing a wide range of network configurations (see Figure 1). Based on the Linux operating system, the cluster offers several parallel computing environments, including PVM, MPI (LAM), TreadMarks and traditional Unix applications with sockets. It is used to test the type of scientific applications run at NIST, as well as to evaluate the performance and the limits of such a platform. One of the clusters was successfully transferred to production in mid-98.

In addition to conventional performance metrics such as system information and software profiling, NIST has developed advanced performance measurement techniques. S-Check [SNE97] is a statistical tool to detect bottlenecks in parallel code. The MultiKron [MIN95] provides a hardware, high-resolution (100ns) tracing facility with very low perturbation –

the cost of a memory write. To use the MultiKron in a distributed environment, hardware instrumentation was developed, using the Global Positioning System (GPS), to synchronize the MultiKron timestamp clocks. The combination achieves global synchronization of all MultiKron boards to within one microsecond worldwide.

The goal of this project is to propose a method to capture an extensive performance profile of a cluster, in order to study its general performance, bottlenecks, possible interactions and many other characteristics. The project focuses on thorough analysis of performance data through extensive reuse of existing data and will propose different analysis and visualization schemes for this specific application. The project is divided into three phases: data collection and storage, data analysis, and run-time toolkit (see Figure 2).

This document discusses the first phase: data collection and storage.

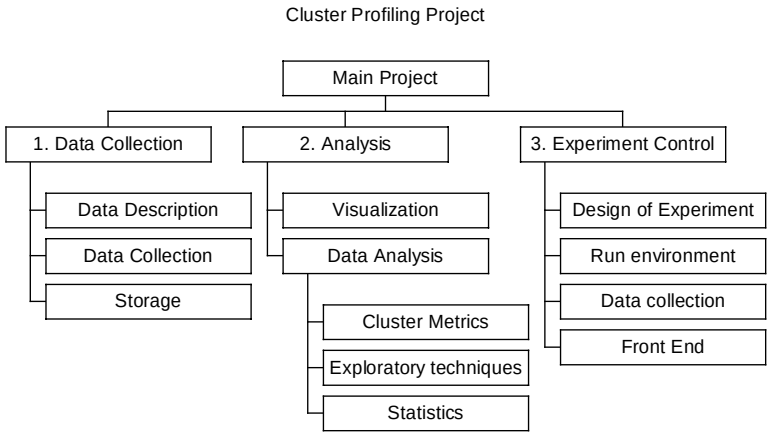

Figure 2. Project Diagram.

1.2 Phase 1

The data collection is divided itself into three tasks.

- Data Abstraction: once an extensive inventory of the known sources of performance data and their properties was prepared, an abstract model was designed to represent the cluster environment in a compact but accurate fashion. This is discussed in section 2.

- Data Collection: after reviewing the tools available to conduct the actual collection of the data from the different sources, a simple reference implementation is provided to demonstrate the collection process as described in section 3.

- Data Storage: once the data has been retrieved from the nodes, it must be used in an efficient and accessible way to promote sample data reuse. Several popular models were evaluated. One solution was selected and implemented, along with basic management and retrieval tools.

1.3 Existing work

Many research groups have focused on some aspects of performance evaluation of parallel computing systems in general and clusters in particular.

Several tools provide advanced application-level tracing specifically designed for parallel software. For example, VAMPIR [VAM99], from the German company Pallas GmbH is an instrumentation and tracing tool for MPI programs. AIMS [YAN96] from NASA Ames Research Center has similar features. MAD [MAD99] from the GUP Linz in Austria provides event-oriented tracing and analysis. Paradyn [MIL95] features performance tracing down to the procedure and statement level through dynamic instrumentation.

Other tools propose visualization techniques, such as Pablo [AYD96] and Paragraph (part of the AIMS package), Paradyn, Upshot or even PARADE [PAR99] from Georgia Tech that also provide advanced navigation tools.

However, as far as we know none of them address the issue of storing, and later accessing, the accumulated performance data. They may use proprietary trace files, or one of the public formats, such as Pablo's SDDF (Self-Defining Data Format), but managing the trace files resulting from multiple experiments is left up to the user.

2 Data Representation

2.1 Overview

The purpose of the collection module is to capture distributed performance snapshots of the cluster environment (network, nodes, other devices, application data), to be combined and stored for later analysis. In this section, we list the major data classes available from the cluster, then we describe a simple, extensible data model to represent each snapshot. We also introduce a higher-level data structure to model experiments.

Extensive instrumentation of a cluster environment can generate a very wide range of data. The following lists some of the most important sources of performance data, which are organized formally later in the document.

- Physical Machine (computer)
  - Machine description (processor, memory, cache size, operating system).
  - Process list (ID, command line, CPU and memory usage, time).
  - Memory information (free, swap, cache misses).
  - Network information (dropped packets, output of tcpdump or netperf).
  - Load.
  - Status and data of MultiKron (clock, flags, timestamps), GPS (time and date, position, status) and other devices.
- Process
  - Process ID
  - Response time
  - Current usage (memory, CPU, swap, interrupt counts, I/O)
- Application trace
  - User-defined trace (usually counters, timers and flags)
  - Compiler options
  - Compiler type
  - Version and General configuration
  - Gprof data: {function name, CPU, I/O times, runtime, call count}
- Communication Libraries
  - Data transfers
  - Number of calls
  - Average message size
  - Current Receiving and Sending parties
  - Timing information
- Network device
  - Configuration
  - Dropped packets
  - ARP Tables
- Virtual machine
  - List of machines
  - Configuration and status
- Queuing System / Resource Manager
  - Configuration
  - Status
  - Individual queue information (waiting time, service time)

2.2 Data Types

2.2.1 Basic Data Types

All the information described above uses any one of these basic data types:

- Integer
- Floating point
- String

Most systems add semantics by:

- Constraining these types, for example small 16-bit integers, signed and unsigned numerical values, limited ranges (such as a percentage, an integer in [0:100]), or even dynamic constraints (for example SNMP counters that can only be incremented);
- Providing arrays of the above;
- Supporting declared subtypes, such as time ticks (unsigned long integers counting tens of milliseconds).
- Supporting compound types such as dates.

2.2.2 Summary Values and Times Series

Two different types of metrics lead to different abstraction models to describe them accurately and efficiently:

- One of the basic or compound data types described above can directly represent single value metrics, or summary data, such as response time or number of messages sent.

- Many metrics however consist of consecutive readings over the course of the experiment and need a special representation for time series. In order to limit the storage burden for very long experiments, we decided to use the Paradyn's data abstraction for time histograms [MIL95]. The number of samples is fixed and the granularity (sampling rate) grows as the experiment runs longer.

> "Paradyn stores performance data internally in a data structure called a time histogram. A time histogram is a fixed-size array whose elements (buckets) store values of a metric for successive time intervals. Two parameters determine the granularity of the data stored in time histograms: initial bucket width (time interval) and number of buckets . . . If a program runs longer than the initial bucket width times the number of buckets, we double the bucket width and re-bucket the previous values. The change in bucket width (time interval) can cause a corresponding change in the sampling rate for performance data, reducing instrumentation overhead. This process repeats each time we fill all

buckets. As a result, the rate of data collection decreases logarithmically, while maintaining a reasonable representation of the metric's time-varying behavior." [2]

2.3 Data Model

A data model is an abstraction that represents reality. It encompasses both abstract data structures and mechanisms to describe the relationships within the data, while leaving out extraneous details. A cluster environment and its sensors generate highly hierarchical and mostly independent data that fits nicely in a tree structure. The data model below also introduces abstract objects such as network nodes or hardware devices. It also makes a distinction between category values (i.e., input values, such as settings, options, which are more or less fixed) and summary values (i.e., output values, such as measured data, usually the target of statistical operations). This classification is not rigid, however, as what one user may consider fixed (e.g. the number of nodes) may not be for another (e.g. for a scalability test).

The cluster model (Figure 4) accurately holds a snapshot of the whole cluster configuration during a measurement session.

The actual data to be stored in the database as a result of an experiment is a set of cluster snapshots (*runs*). An extra layer of abstraction (Figure 3) is then required to group these runs into a logical structure that describes the *experiment*. This powerful abstraction provides the ability to design *virtual experiments* after the fact by recombining existing runs. This is the key to reusable performance data.

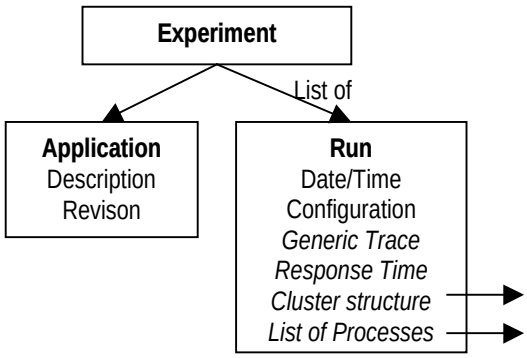

Figure 3. Experiment Model.

[2] Quote from [MIL95].

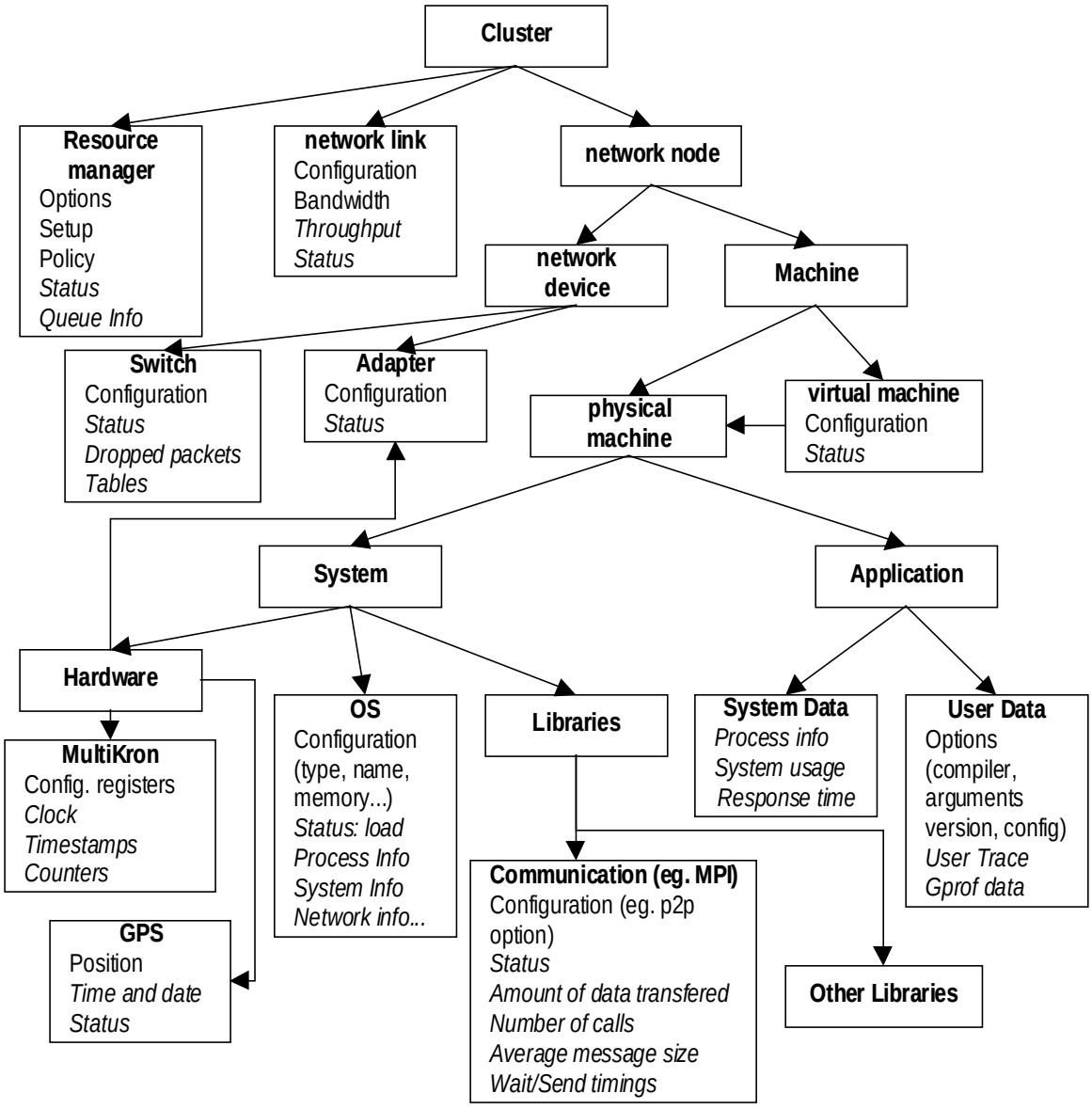

Figure 4. Cluster Model.

3 Data collection

Data collection being one of the main parts of the project, we present in this section different options, from the simplest procedures followed when running an experiment by hand to more complex industrial grade protocols.

3.1 Mechanisms

In this part, we review different mechanisms available to retrieve data from the sources of our data model (Figures 3 and 4).

3.1.1 No specific mechanism

This is the simplest approach. It consists of little shell scripts using the Telnet protocol and directly executing commands (displaying information or results on screen or in a file) or using the FTP protocol to retrieve files. This method needs a lot of managing and parsing of files manually. However, it does not require a lot of initial effort. We can use it quickly without preparation but for each new experiment, we have to start over in the creation of new scripts. In addition, it is not easily extensible.

3.1.2 Client-server architecture

3.1.2.1 Rsh

Rsh (remote shell) is a protocol available under Unix and other platforms that allows remote execution of commands or shell scripts on a machine. A background process (daemon) runs on each host to process the requests. In our case, we can use it to retrieve information from different machines by running commands the same way as above, but remotely. This method is readily available on all Unix systems and easy to implement. However, a major drawback for this mechanism is that a user must have full access to all the machines. This is obviously a security problem. Moreover, on some architectures, including the version of Linux running on the cluster, the rsh daemon tends to hang if called too many times or too often.

3.1.2.2 Custom-made server

This option requires a lot of initial effort although it is fairly easy to develop. It requires the development of two applications:

- a server application running on each node;
- a client application that gathers data from each server.

Since everything is designed specifically for this application, the server can provide all the required functions in a compact and efficient way. This option leads to a generic and extensible tool. There are no extraneous features, reducing the server footprint (and therefore potential perturbation) to a minimum. In addition, the server may implement security mechanisms, such as passwords or encryption, to protect the data and restrict access. It will also probably perform better than the mechanisms described earlier since all the transactions can be performed over a single connection. However, some nodes (e.g. a switch) cannot run such a server. Moreover, interoperability requires some form of network

standard data representation, such as ASN1 or XDR to transfer the data, which adds to the initial development cost.

3.1.2.3 Standard interfaces

Standard interfaces offer accrued interoperability for a fraction of the initial development cost of a custom-made server.

FTP

The File Transfer Protocol provides easy data transfer with a basic security scheme (username, password and limited file access). This protocol supports multiple simultaneous transfers in during the same session allowing faster retrieval of files; there is no need to repeatedly open and close new sessions, like with rsh. However, FTP is not available on all the nodes although some network devices do support TFTP, a simple non-secure alternative to FTP.

All transfers are file-oriented, so an FTP-based collection device has access to:

- The results of Unix commands or combination of those using “pipes”, this solution leads nearly to the “rsh” one without its security issue.
- The information found in the /proc tree of some Unix systems. They provide direct access to the operating system, giving general information about the machine and various kernel modules.
- Files created and possibly updated by a running daemon on the remote machine. This option adds extensibility and is quite simple to implement.

FTP itself is a passive solution: it can only retrieve files. Their content must be updated by other agents, such as the kernel, a daemon or an application.

HTTP

The Hyper Text Transfer Protocol, used in web sites, gives the same features as FTP in terms of file transfer but with limited security options. On the other hand, it gives access to active content using CGI scripts or Java applets. The data can then be stored as files and/or displayed (using HTML tags) on any machine with a web browser. Many modern network devices, for example our Ethernet switch, have an HTTP interface.

SNMP

The Simple Network Management Protocol gives the user the capability to manage a remote network node by setting values and monitoring network events. It uses:

- Several agents, which are applications running on each machine to be monitored or managed.
- One or several managers that connect to the agents to monitor and manage the devices by getting and setting properties and listening to events.

The properties are data variables representing resources on the SMNP node handled by this agent. The variables are organized in an MIB (Management Information Base), a kind of dictionary ordering them in a tree structure (the MIB tree) that defines them (name, type and description) and arranges them in a logical way.

SNMP provides three main types of requests:

- GET requests to retrieve values from the agent.
- SET requests to update values on the agent.
- TRAP messages from an agent to notify the management station of significant events that have occurred.

Implementing the SNMP standard requires a significant initial development effort (detailed protocol, complex data representation), but there are many tools available in the form of libraries, generic agents and managers, both free and commercial. More importantly, many network devices, such as switches and router, support SNMP management. This gives SNMP the greatest outreach of all the mechanisms evaluated in this document.

The cluster application uses only a limited subset of the features provided by SNMP – traps and setting values are not used. Most agents available on the market do implement all the functionalities, which can lead to a very large agent in memory.

Security is a weakness of the protocol. In the initial version of SNMP, all communicating agents are assigned two community strings (one to read and one to write data), as a makeshift for passwords. These strings are sent unencrypted over the network. Virtually anyone can get information and change the configuration of the SNMP nodes. SNMPv2, the second version of the protocol, specifies an authentication service using DES encryption.

3.1.3 Summary

Table 1 below rates different features important to us for each mechanism described above:

X	Basic solution	Rsh	Custom made server	FTP	HTTP	SNMP
Extensibility	■	■■	■■	■■	■■	■■■
Security	■■	■	■■■	■■■	■	■
Performance	■	■	■■■	■■■	■■	■■■
Maintenance	■	■	■■	■■	■■■	■■
Size	■■■	■■■	■■	■■■	■■■	■■
Complexity of client side	■	■■■	■■■	■■■	■■■	■■■
Complexity of server side	■■■	■	■	■	■	■
Initial development effort	■■■	■■	■	■■	■■	■■
Tools available	■	■	■	■■	■■■	■■■
Access to all nodes	■■	■	■	■	■	■■■
Interoperability	N/A	■	■	■■■	■■■	■■■

Table 1. Ranking of Various Collection Mechanisms (■=poor – ■■■=good).

Based on this table, we selected SNMP because:

- SNMP is a standard, thus it is available on computers as well as on other network nodes such as switches or routers.
- It is extensible.
- As a standard, it should be available for new hardware in the future.
- Many SNMP tools are already available, reducing the amount of work needed.

3.2 Sample implementation

Once the choice of SNMP was made, we looked for some available agents and managers.

3.2.1 SNMP Libraries

We first found SNMX from New Line Software, Inc. [ACE99], a free SNMP package using scripting language. It was possible to make small programs, but the source code was not available. Thus, extensibility of the agent was difficult to achieve.

Next, we investigated Carnegie Mellon University SNMP package (CMU-SNMP) [CMU99]. This free software consists of a library only. It is highly experimental and comes very limited documentation. However, source code was available for modification.

We then investigated the University of California at Davis SNMP package (UCD-SNMP) [UCD99]. This is an improved version of CMU-SNMP that is enhanced and supported. It comes with utilities performing several SNMP operations. The package is free; source code and detailed documentation are available, making it easily extensible. We selected this package for the reference implementation.

The agent of the package was our focus since we want to augment it to implement new sensors. Written in C language, its modular structure allows new sensing modules, using the provided templates.

3.2.2 SNMP Manager

As an added feature to using SNMP for our collection infrastructure, any SNMP manager can monitor and manage the cluster. Most managers have a graphical interface to display the status and properties of each machine on one single screen. This feature was not an essential part of the project but could help in managing the machines.

The UCD-SNMP package provides a robust and easily extensible agent but only a command-line manager. We then tried to look for a graphical SNMP manager with customization possible for new sensors on the agents. We evaluated several commercial managers with a graphical interface and customization capability to support the new attributes:

- Net Inspector from MG-SOFT: no customization possible;

- Novell Managewise: overly NetWare-centric;

- SNMPc from Castle Rock is quite customizable but lacks some simple features such as updating at a given rate;

- Tkined from the University of Twente, The Netherlands, is a highly customizable Tcl/Tk network editor based on Scotty, an SNMP extension to Tcl.

We used both Tkined and SNMPc to monitor the network and a Java SNMP API from AdventNet, Inc. to write portable Java applets and applications.

4 Storage

Before describing different models available to support the cluster profiling tool, here are the general requirements of any scientific database. A good database system should provide:

1) A data model providing the abstract representation of the data structures and semantics.
2) A high-level query language to easily access the data.
3) Support for concurrent access by multiple users.
4) Mechanisms validating the integrity of the data.
5) Safe recovery from possible system failures.
6) Efficiency.

In this section, we will propose several database models: a flat-file storage system, an object-oriented model, a relational model and the OLAP approach to it. We will also propose a sample implementation with a traditional relational database system.

4.1 Database Survey

4.1.1 Naïve solutions

The simplest database system possible is a set of flat files, each one holding the data from one or several runs, or a subset of it. Recording and storing the performance profile are extremely easy with a simple record-oriented format. Accessing the records, however, is very difficult: even if the record format is well designed, querying the database requires appropriate tools and is most likely to be very slow. Tedious management (removing, sorting, adding records, etc.) and limited extension capabilities make such a simplistic database quite impractical for a large number of measured experiments.

There are ways to improve this solution.

- A richer file format with embedded structure description can improve data availability. Possible candidates are the Self-Defining Data Format (SDDF) [AYD96] or open standards such as SGML or its subset XML.

- A simple Web site can offer an easy solution to the query issue, by hosting all the data in a standard format (most likely HTML or XML). The site then runs a search engine (e.g. Harvest [HAR96], which supports SGML, HTML and RTF) to index and then help to select records. As described in the Harvest documentation, using meta-data (such as the META tags in HTML) can substantially improve the efficiency and accuracy of the queries. This solution, although somewhat inefficient, is very easy to implement and can lead to fairly interesting results.

4.1.2 Object-Oriented Database Management Systems (OODBMS)

Object-oriented concepts provide the tools to design a rich data model that naturally matches the structure of data, as well as the abstraction power of inheritance and encapsulation. Object models offer a uniform treatment of a wide variety of data types, including highly complex objects, beyond the standard types available to a traditional database.

4.1.2.1 Data Model

Our model augments the schema described in the previous section (Figures 3 and 4) by extending the basic hierarchical structure with strong object-oriented concepts, as illustrated in Figure 5.

The cluster is viewed as a set of interconnected nodes. The node class can then be subclassed into a network device (a switch for instance) or a computational node. Note that an Ethernet bus and a token ring share the same structure as a switch. Similarly, we use an abstract "Machine" class that can contain other machines, which is then subclassed into a virtual and physical machine. This schema easily and elegantly describes a virtual machine such as PVM, that runs on top of a set of actual workstations. On the other hand, a Java virtual machine running on a workstation fits in this model too. Similar abstract classes organize several other objects in the system.

4.1.2.2 Available OODBMS

Among the main vendors of object technology, Object Design is the leader with 38% of the object database market, followed by Versant and Objectivity. Softwarebuero distributes an all-Java Linux-based object database, free for non-commercial use. Table 2 shows a the main solutions available.

Vendor	**Product**	**Price**	**Website**
softwarebuero m&b	Ozone	Free	http://www.softwarebuero.de
Object Design, Inc.	ObjectStore PSE/Pro	N/A $245	http://www.odi.com
Objectivity	Objectivity/DB	N/A	http://www.objectivity.com
UniSQL	UniSQL/X	N/A	http://www.unisql.com
Versant Object Technology	Versant	N/A	http://www.versant.com
Ardent Software Inc.	O_2	$5000	http://www.vmark.com/object
Poet Software	Poet	N/A	http://www.poet.com

Table 2. OODBMS Survey.

4.1.2.3 Adequacy

The object model matches the natural structure of data so well that it makes storing the performance traces collected from the cluster a very simple procedure, as virtually no translation is needed. An OODBMS is probably quite efficient, for the same reasons.

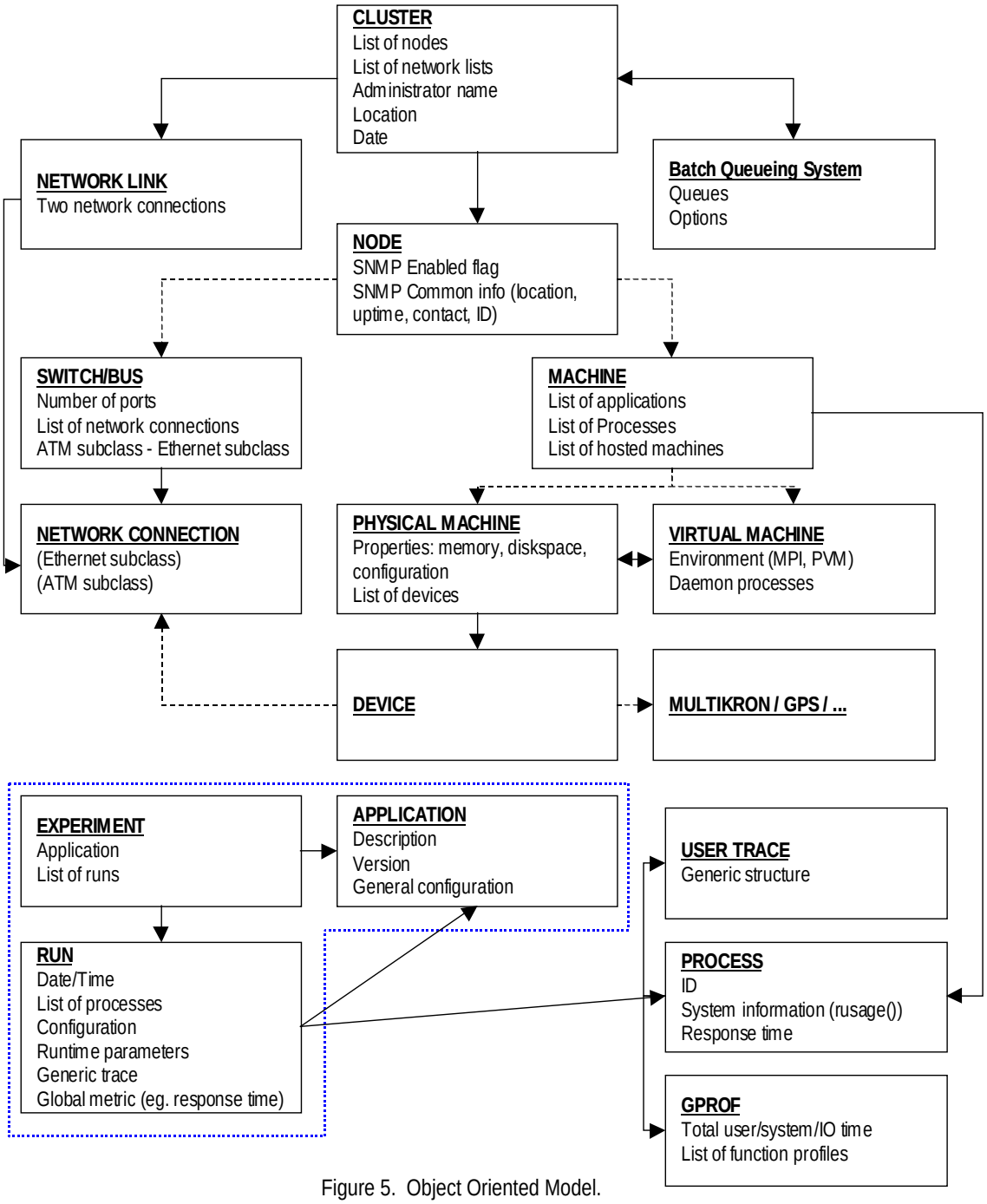

Figure 5. Object Oriented Model.

However, free or open OODBMS are still at an experimental stage and somewhat unstable, while commercial systems are quite expensive. But the main weakness of OODBMS available today is the lack of a standard query language. In spite of the efforts of the ODMG [ODM98] for example, who developed OQL (Object Query Language) an extension to SQL92, very few systems were OQL-enabled (at this time, POET and ObjectStore Server).

4.1.2.4 Example with ObjectStore

ObjectStore PSE Pro for Java from Object Design, Inc. is a Java API that provides mechanisms to create, manage and connect to object oriented databases.

We first need to create the objects. The mapping between the above object model and Java is straightforward. For instance, here is a sample declaration of the machine object:

```
/* Machine.class */

public class Machine extends node {
  private String name;
  private Process processes[ ];

  public Machine(String name, Process processes[ ]) {
    this.name = name;
    this.processes = processes;
  }

  public Machine() {
    this.name = "";
    this.processes = null;
  }

  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public Process[ ] getProcesses() { return processes; }
  public void setProcesses(Process processes[ ]) { this.processes = processes;}
  public String toString() {
    String s = "Machine:\n" + "  name = " + name + "\n";
    s += "  processes: " + processes.length + "\n";
    for (int i=0; i<processes.length || i==1; i++)
      s += processes[i].toString();
    return s;
  }
}
```

We then need to create an instance of the object. Once the instance is created, we can easily insert data as objects in the database. Indeed, by inserting the top object of the structure, the API inserts all objects reachable from this top object. However, this ease of insertion has a downside: we can query only the top object, i.e. the cluster in our example. There is no way, with ObjectStore PSE Pro, to query an object under (in the structure) the top object without recursively searching the children “by hand”. This is not satisfactory in a context where querying is very important.

4.1.3 Relational Database Management Systems (RDBMS)

Relational databases are the most popular in the database world. Contrary to object oriented databases where all the data are well structured and logically bound together, relational databases rely on several tables linked with relations.

4.1.3.1 Model

Description

We used an Entity-Relationship Diagram (ERD) (see [OCC92]) to represent a simplified view (Figure 6) of a possible implementation of a relational database for this application. Each box represents a table, the basic entity type in the relational model. Each link between two boxes represents a logical relation between the data in the tables. Three types of relation exist:

- (n:n) “many to many” and not used in our example.
- (1:n) “one to many” which is used for example between an experiment and a run. This is the relation used to represent a list. Indeed, here, an experiment is a list of several runs and one run can belong to only one experiment.
- (1:1) “one to one” relation meaning that, for instance, each machine can have only one Multikron device and that each Multikron device belongs to only one machine.

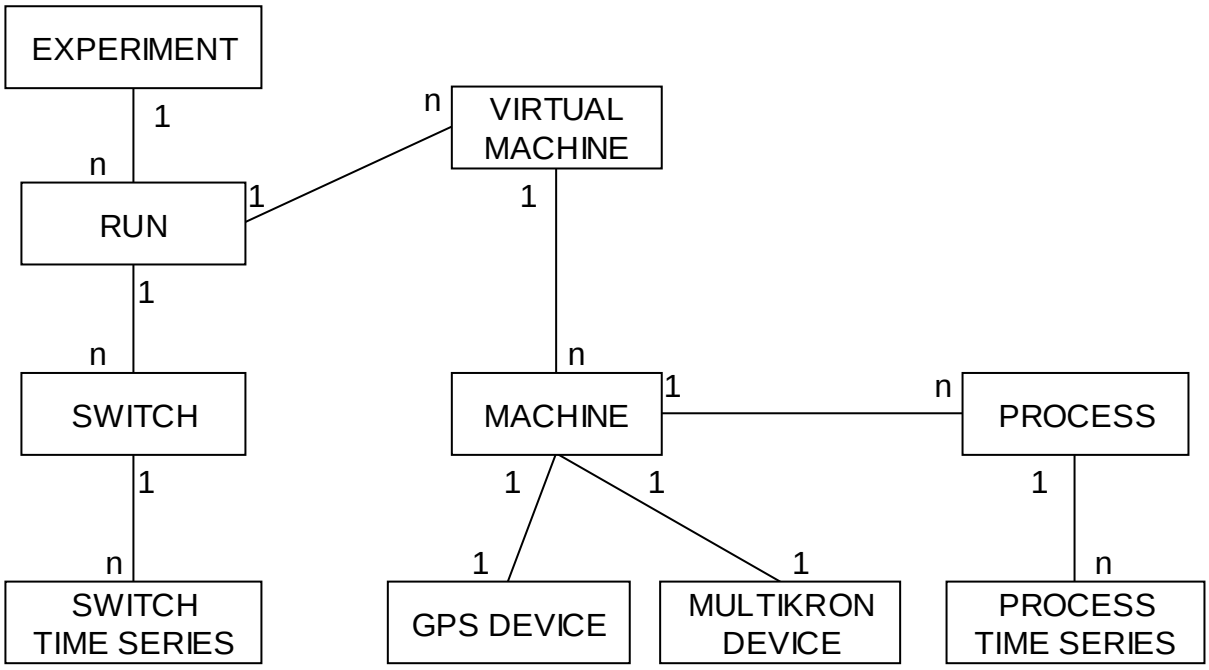

Figure 6. Entity-Relationship Diagram.

All those different relations have some consequences in the building of the tables, especially the primary keys (see [OCC92]), which are attributes of tables that make each entity occurrence unique.

Pros/Cons, performance

The relational model is a mature technology, one of the first ones and still widely used. It is simple, efficient and, in general, less expensive than object oriented technology.

This model, contrary to the object-oriented scheme, does not reflect the data structure. This fact leads to two kinds of complexity:

- Complexity at design-time when all the relations and tables have to be defined. As we will see in the sample implementation section, the Entity-Relationship model of figure 6 leads to a complex primary key management. However, this burden is on the database designer only.

- Complexity in querying since we will need to make a lot of joins and projections (operations on tables, see [GRU90]) on large tables, resulting in poor performance for large databases. The database management system handles most of the work. The consequence is that querying, which is, in our case, very important for the analysis of stored data, is fairly easy and flexible for the end-user, even though a lot of relations may appear in the data model.

In addition, the primary key management involves some redundancy in the different tables, as we will see in the reference implementation.

4.1.3.2 Available database solutions

Many relational database system products are inexpensive or even free (for non-commercial products). We looked for Linux based software.

Commercial products provide, in general, many unneeded features. They are usually more affordable than object oriented solutions, but still expensive. Some free products seem to be sufficient however

Some relational databases are [SQL98]:

- mSQL (MiniSQL), commercial, inexpensive; its query language is a very limited subset of SQL.

- GNU SQL, free (GNU license), still in early development.

- BeagleSQL, free, very much in development, not stable.

- PostgreSQL, free, stable, simple.

- MySQL, free extension of MiniSQL, simple of use.

4.1.4 OLAP and Multi-Dimensional Databases (MDBMS)

OLAP (On-Line Analytical Processing) is a database software technology that enables analysts to view and access a traditional database as a multi-dimensional structure. Tables 3 and 4 demonstrate how a traditional relation can be viewed as a multi-dimensional structure.

Salesperson	Product	Sale Amount
Bob	Nuts	2000
Mary	Bolts	6000
Bob	Bolts	3000

Table 3. Traditional Relation.

X	Nuts	Bolts
Bob	2000	3000
Mary	0	6000

Table 4. Multi-Dimensional View.

This model, along with the underlying implementation and accompanying tools, allows for fast, natural and powerful data analysis along all the dimensions. It is becoming popular for high-volume trend analysis and data mining of large-scale databases. The data model also includes hierarchies within dimensions and multi-dimensional vector arithmetic and tools (including extensions to SQL). OLAP servers often support time-series (time being one dimension), a very useful tool for trend analysis.

OLAP may be the architecture for the next generation of high-performance business databases. It can be implemented on top of traditional relational (or other) database management systems, although with poorer performance.

OLAP is quite appealing for many applications. Such systems are designed – and priced – for high-volume industrial applications.

4.1.5 Database Access

There exist several standard ways to access a DBMS (Figure 7), all of them supported to some extent by leading database system vendors.

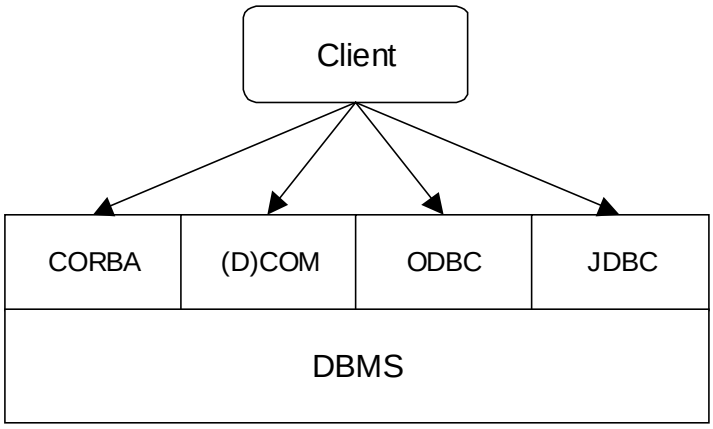

Figure 7. Standard Access Methods to a DBMS.

4.1.5.1 CORBA

The Common Object Request Broker Architecture (CORBA), developed by the Object Management Group [OMG99], is an open, distributed object infrastructure that allows applications to communicate through a standard object interface provided by an Object Request Broker (ORB). The OMG describes the ORB as “the middleware that establishes the client-server relationships between objects. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its method, and return the results. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an object's interface. In so doing, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.” Figure 8 shows the structure of a CORBA-based system.

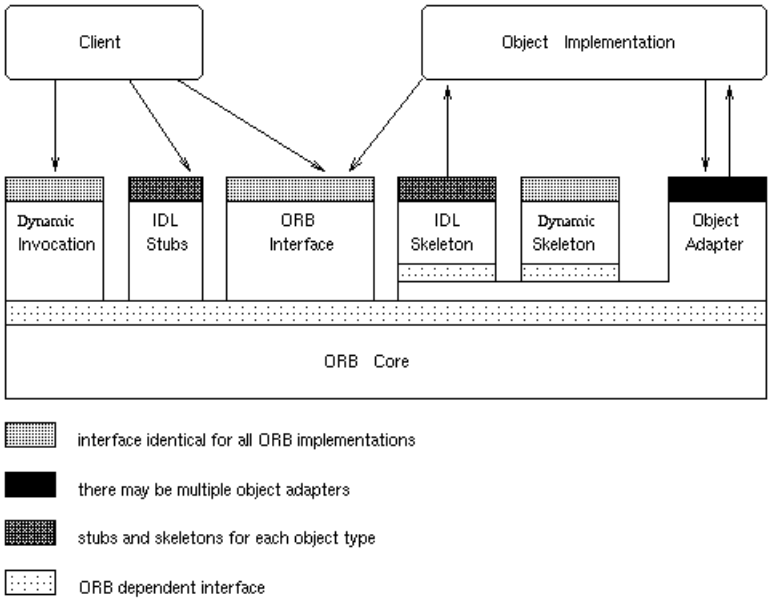

Figure 8. CORBA Architecture. [KEA97]

There are no free fully compliant CORBA solution but many commercial implementations are available.

The object-oriented model described in Figure 5 may be implemented with CORBA, although this architecture was designed for distributed and heterogeneous systems. For this application, it adds little benefit to a local Java or C++-based object broker, except that the ORB manages the objects similarly to an object-oriented database. As before, querying, which is essential, is left up to the object implementation.

CORBA, as well as Microsoft’s COM [COM99], provides essential distributed object services but still requires an underlying data management system. Major commercial databases have a CORBA interface.

4.1.5.2 ODBC and JDBC

JavaSoft's JDBC and Microsoft's ODBC (Open DataBase Connectivity) are both APIs for executing SQL queries. They provide a standard interface to nearly any database on the market. Microsoft's ODBC is the most widely used database drivers but JDBC integrates naturally into Java – a JDBC-ODBC bridge is available for those database systems that do not support JDBC.

Example of JDBC query, from the SunSoft documentation:

```
Connection con = DriverManager.getConnection(
    "jdbc:odbc:wombat", "login", "password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b FROM Table1");
while(rs.next()) {
     int x = rs.getInt("a");
     String s = rs.getString("b");
}
```

4.2 Reference Implementation

4.2.1 General description

Our choice to implement the data storage solution is the relational model. After having encountered querying problems with object oriented databases, technology that seemed to be best suited for what we wanted to do, we started evaluating some relational database management systems. Our priorities were reuse, access and easy retrieval of stored data. These features were all present in relational database systems whereas the querying in object oriented database systems was not satisfactory. Multidimensional databases are business-strength solutions that do not suit smaller-scale application.

Our current implementation uses PostgreSQL [POS99] running under the Linux operating system, arguably the most popular and complete free DBMS available. PostgreSQL supports most of the SQL constructs, including sub-queries, user-defined types and functions.

A named table is the representation of an entity type in the relational model. Table columns represent attributes of the entity type and each row in the table corresponds to an entity occurrence. The relational model requires that each entity occurrence of each table be unique. This uniqueness is achieved using primary keys;they are embedded in other tables to make each entity occurrence unique and to achieve cross-references between tables.

We had to translate the entity-relationship diagram (Figure 6) into an actual database. The sample result, which is not yet complete, is shown in Figure 9.

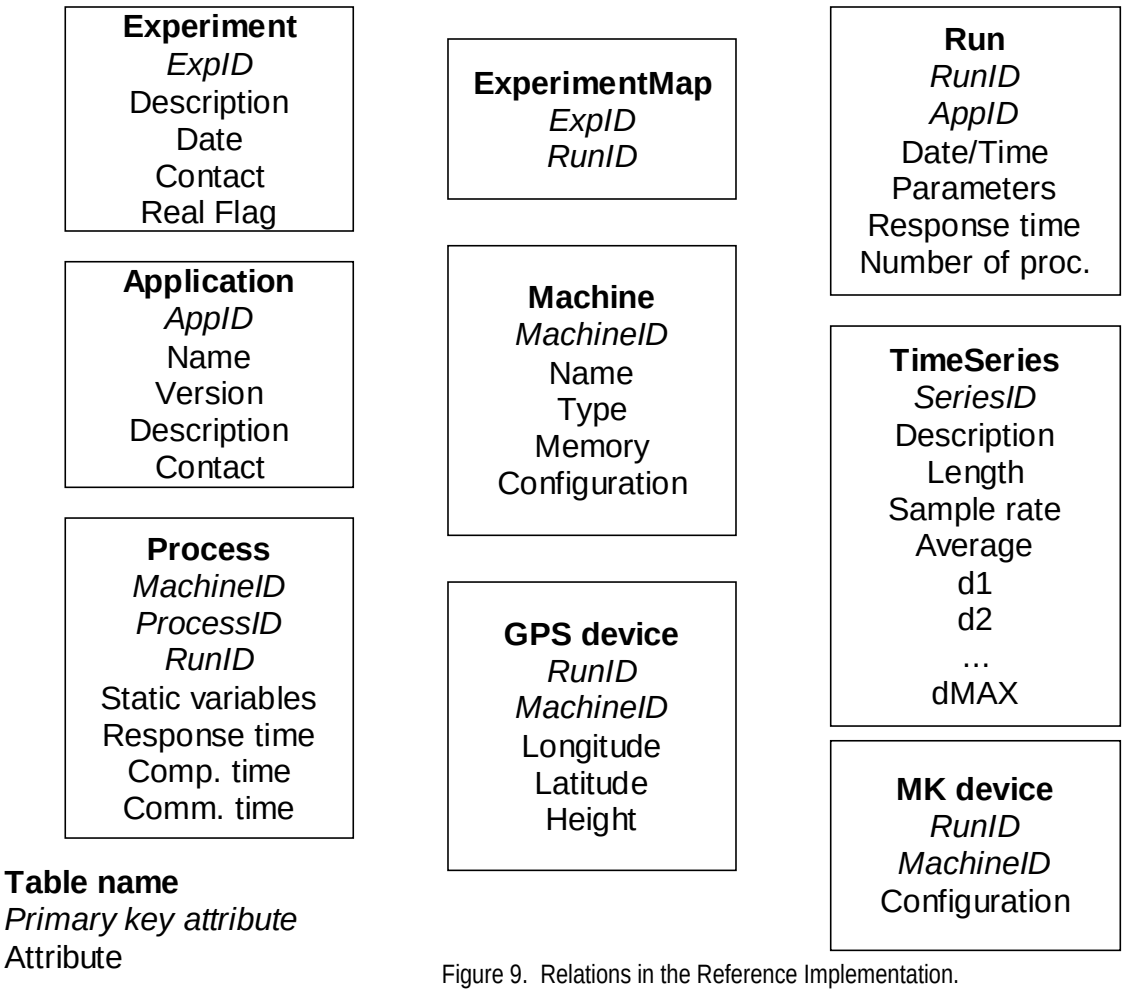

Figure 9. Relations in the Reference Implementation.

The type of data is commonly integer, float or string. We notice the redundancy of the model. For each “one to many” relation, the primary key of the content is made of the primary key of the container and a unique attribute of the content itself. For each “one to one” relation, the primary key of the container is the key of the content itself. These are ways to ensure the uniqueness of each record.

Each table in Figure 9 contains attributes: some are parts of the primary key of the table and some are actual data. Some data needs a special type of storage – time series – so we can analyze variations of those data with time. A possible way to implement these time series is to use a specific table for each type (integer, float, string) of data and for each table that may require time series (Process, Multikron device). This table is then filled the same way that Paradyn [MIL95] fills its time histograms (see above), so that we can store a trace of a long or short experiment using the same amount of memory. In addition, this approach provides, in one single query, all interesting information to plot graphs. This is a significant feature for our application.

4.2.2 Customization

Each kind of performance experiment needs some specialized data, which can be of different types (integer, float and string). To avoid creating one table for each different type, we chose to create one table that included a key name and three attributes for integer, float and string values. This facilitates the access to data. One attribute out of three is used, the others being set to NULL. Three generic registries were added to supplement existing run, machine and process information (Figure 10).

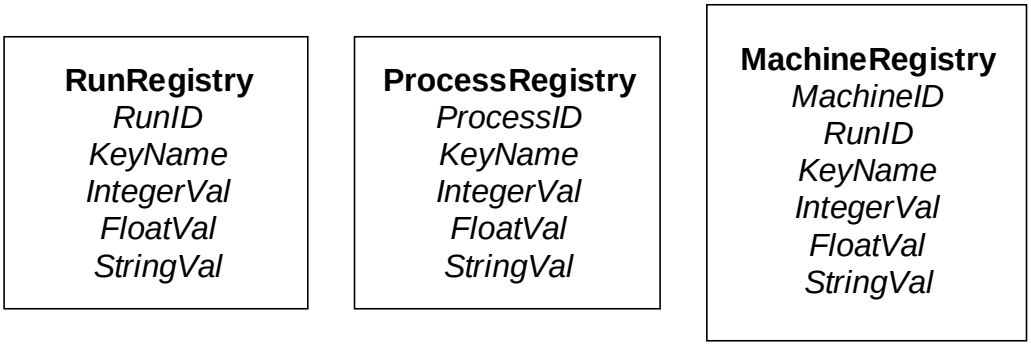

Figure 10. Generic Registries for Run, Process and Machine-Specific Data.

4.2.3 Example of SQL queries

Using Time Series

As an example, we have stored data from an experiment of 4 runs, consisting of 1,2,4 and 8 machines with one process on each machine (there can be other processes running on each machine). In addition, we have gathered the memory usage as a process time series. We want to query the average memory usage for each process of each run. The simple SQL query below provides this information:

```
Select P.RunID, P.ProcessID, PTS.Average                    what we want to retrieve
from Process as P, ProcessRegistry as PR, TimeSeries as PTS  tables part of the query
where P.ProcessID = PR.ProcessID AND P.ProcessID = PTS.ProcessID  joins between tables
AND PR.intvalue = PTS.SeriesID AND PR.KeyName = 'MemoryUsage';
```

The result of this query yields 15 different average values. We can see that the database is not too laborious to query although it might require significant work by the database application for large tables.

Effect of Optimization in Compilers

We have stored extra configuration data in the RunRegistry for each run, such as the number of processes and the level of compiler optimization (e.g., -O2 or nothing with gcc). We want to compare the average response time of all those runs where the number of processes is fixed. To accomplish this, we make two queries, retrieving the average response time with and without optimization as shown below.

```
Select avg(ResponseTime)                                    selection of what we want
from Run as R, RunRegistry as RR1, RunRegistry as RR2       tables part of the query
where R.RunID = RR1.RunID AND R.RunID = RR2.RunID AND       joins between tables
                                                            selection of application
RR1.KeyName = 'numProcessors' AND RR1.IntegerVal = 16       selection of number of processes = 16
RR2.KeyName = 'optimization' AND RR2.StringVal = '-O2'      selection of optimization
```

This query should return a single value, the desired average. The other query is the same except the last clause: RR2.StringVal = ''.

4.2.4 Connection to the database server

PostgreSQL provides a wide range of access methods in addition to the standard command-line SQL interface: C, C++, PERL and Tcl bindings, embedded SQL in C and OBDC and JDBC interface. JDBC (Java Database Connectivity) and ODBC (Open Data Base Connectivity) offer a standard method to access virtually all relational database servers (in case we need to change the DBMS, the source codes will stay the same). The graphical interface that comes with PostgreSQL (pgaccess, Figure 12) allows for convenient and simple browsing of the database from most platforms.

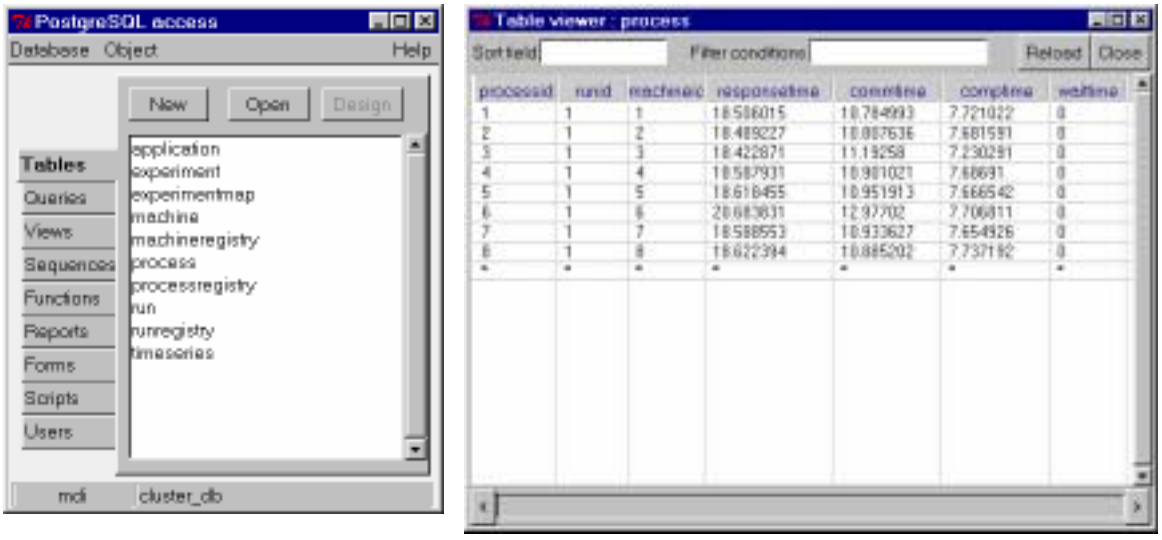

Figure 11. Screenshots of Pgaccess.

5 Summary

First we established an inventory of the data available on a cluster, then we reviewed different collection techniques. After evaluation, we believe that Simple Network Management Protocol (SNMP) has the most potential to give access to the most profiling information in a cluster environment in a unified fashion.

Organized storage of performance data required the selection of a database technology. Although the object-oriented model seemed to be well suited to our needs, the tools implementing the technology are not mature enough. We selected the relational model and implemented our storage and collection engine around PostgreSQL, a robust freeware RDBMS for Linux.

The data collection and storage phase of the project is now almost complete. Our team is currently working on the second and third phases of the project:

- Visualization of the collected data, as well as statistical methods and metrics specific to the performance analysis of cluster computing (data analysis and visualization module).

- Simplify and integrate the setup, execution and storage of experiments on parallel clusters (design of experiment and run-time module).

References

[ACE99] Diversified Data Resources, Inc., “ACE-SNMX Scripting Executive” http://www.ddri.com/acesnmp/snmx_info.htm, 1999.

[AYD96] Ruth A. Aydt, “The Pablo Self-Defining Data Format”, http://www.pablo.cs.uiuc.edu, 1996.

[BAK57] Seán Baker, “CORBA Distributed Objects”, *ACM Press*, 1997.

[BAN92] François Bancilhon, Claude Delobel, Paris Kanellakis, “Building an Object Oriented Database, the O_2 experience”, *Morgan Kaufmann Publishers*, 1992.

[CAT91] RGG Catell, “Object Data Management”, *Addison-Wesley*, 1991.

[CAT97] R.G.G. Catell et al., “The Object Database Standard: ODMG 2.0”, *Morgan Kaufmann Publishers*, 1997.

[CMU99] Carnegie Mellon University, “CMU SNMP Library”, http://www.net.cmu.edu/projects/snmp, 1999

[COM99] Microsoft, “Microsoft COM, Component Object Model”, http://www.microsoft.com/com, 1998.

[DAT96] “OLAP And OLAP Server Definitions”, *Datamation, April 15, 1996*.

[GRU90] Martin Gruber, “Understanding SQL”, *Sybex*, 1990.

[HAR96] Harvest Web Indexing, http://www.tardis.ed.ac.uk/harvest.

[KEA97] Kate Keahey, “A Brief Tutorial on CORBA”, http://www.cs.indiana.edu/hyplan/kksiazek/tuto.html, 1997.

[KIM96] Ralph Kimball, “The Data Warehouse Toolkit”, *John Wiley and Sons, Inc.*, 1996.

[MAD99] MAD, Monitoring And Debugging environment, http://www.gup.uni-linz.ac.at:8001/research/debugging/mad/, 1999.

[MIL95] B.P. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Irvin, K. Karavanic, K. Kunchithapadam and T. Newhall, “The Paradyn Parallel Performance Measurement Tools”, *IEEE Computer, 28*, 1995.

[MIN95] Alan Mink, “Operating Principles of Multikron Virtual Counter Performance Instrumentation for MIMD Computers”, *NISTIR 5743,* November 1995. http://www.multikron.nist.gov.

[MYS99] T.c.X. DataKonsultAB, “MySQL”, http://www.tcx.se/, 1999.

[OCC92] Val Occardi, “Relational Databases: Theory and Practice”, *NCC Blackwell*, 1992.

[OMG99] The Object Management Group, http://www.omg.org, 1999.

[PAR99] Georgia Institute of Technology, “Visualization of Parallel and Distributed Programs”, http://www.cc.gatech.edu/gvu/softviz/parviz/parviz.html, 1999.

[POS99] “PostgreSQL Home Page”, http://www.pyrenet.fr/postgresql/, 1999.

[SQL98] Linas Vepstas, “Linux SQL Databases and Tools”, http://linas.org/linux/db.html, 1998.

[SNE97] Robert Snelick, Mike Indovina, Michel Courson, Anthony Kearsley “Tuning Parallel and Networked Programs with S-Check”, *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97), Las Vegas, Nevada* (June 30 - July 3, 1997), http://cmr.ncsl.nist.gov/scheck, 1997

[UCD99] University of California at Davis, “The UCD-SNMP project home page”, http://www.ece.ucdavis.edu/ucd-snmp, 1999.

[VAL89] Patrick Valduriez, Georges Gardarin, “Analysis and Comparison of Relational Database Systems”, *Addison-Wesley*, 1989.

[VAM99] Cornell Theory Center, “VAMPIR”, http://www.tc.cornell.edu/UserDoc/Software/PTools/vampir/, 1999.

[YAN96] J. C. Yan and S. R. Sarukkai, “Analyzing Parallel Program Performance Using Normalized Performance Indices and Trace Transformation Techniques”, *Parallel Computing* Vol. 22, No. 9, November 1996. pages 1215-1237, 1996

[ZBI91] Zbigniew Michalewicz, “Statistical and Scientific Databases”, *Ellis Horwood*, 1991.